

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ  
к лабораторной работе №2  
на тему

**СИММЕТРИЧНАЯ КРИПТОГРАФИЯ. СТБ 34.101.31-2011**

Выполнил: студент гр.253502 Канавальчик А.Д.

Проверил: ассистент кафедры информатики  
Герчик А.В.

Минск 2025

## СОДЕРЖАНИЕ

1 Цель работы .....	3
2 Теоретические сведения.....	4
3 Ход работы .....	6
Заключение.. .....	8
Приложение А (обязательное) Листинг программного кода.....	9
Приложение Б (обязательное) Блок-схема алгоритма, реализующего программное средство .....	14

# 1 ЦЕЛЬ РАБОТЫ

Цель данной работы – изучить теоретические основы и разработать программную реализацию алгоритмов симметричного шифрования в соответствии со стандартом СТБ 34.101.31-2011, обеспечивающих конфиденциальность данных в режиме простой замены.

В ходе работы предстоит решить следующие задачи:

1 Изучить теоретические основы стандарта СТБ 34.101.31-2011:

- Проанализировать структуру базового алгоритма шифрования, включая преобразование данных, раундовую структуру и используемые криптографические примитивы.

- Исследовать математический аппарат алгоритма: арифметические операции по модулю  $2^{32}$ , побитовые преобразования, нелинейные подстановки.

- Изучить особенности режима простой замены и его криптографические свойства.

2 Разработать программную реализацию алгоритма на языке *Python*:

- Создать модуль шифрования и дешифрования данных, реализующий полный цикл преобразований согласно стандарту.

- Реализовать систему управления ключевой информацией с поддержкой 256-битных ключей и возможностью использования пользовательских ключевых данных.

- Внедрить механизмы обработки данных: выравнивание блоков, дополнение до кратного размера, обработка неполных блоков.

3 Протестировать корректность работы реализации:

- Проверить корректность шифрования и дешифрования на различных тестовых наборах данных.

## 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Симметричная криптография представляет собой метод шифрования, при котором для операций зашифрования и расшифрования применяется один и тот же секретный ключ. Данный подход характеризуется высокой скоростью обработки информации и активно используется для обеспечения конфиденциальности данных.

СТБ 34.101.31-2011 является белорусским стандартом симметричного шифрования, устанавливающим единые криптографические алгоритмы для защиты информации. Стандарт определяет семейство криптографических преобразований, построенных на основе базового алгоритма шифрования блока данных. Этот стандарт разработан для обеспечения современных требований к безопасности информации и соответствует международным криптографическим требованиям.

Основные принципы алгоритма включают использование 128-битного блока данных и ключа шифрования длиной 256 бит. Алгоритм реализует итеративную структуру преобразований с выполнением 8 раундов. Каждый раунд включает операции сложения по модулю  $2^{32}$ , побитовые логические операции XOR, нелинейные преобразования с использованием S-блоков и циклические сдвиги битовых последовательностей. Такая комбинация операций обеспечивает высокую криптостойкость алгоритма.

Классификация алгоритмов стандарта включает восемь групп:

- 1) алгоритмы шифрования в режиме простой замены;
- 2) алгоритмы шифрования в режиме сцепления блоков;
- 3) алгоритмы шифрования в режиме гаммирования с обратной связью;
- 4) алгоритмы шифрования в режиме счетчика;
- 5) алгоритм выработки имитовставки;
- 6) алгоритмы одновременного шифрования и имитозащиты данных;
- 7) алгоритмы одновременного шифрования и имитозащиты ключа;
- 8) алгоритм хеширования.

Первые четыре группы предназначены для обеспечения конфиденциальности сообщений. Режим простой замены обеспечивает независимое шифрование блоков данных. Режим сцепления блоков реализует шифрование с зависимостью от предыдущего блока. Режим гаммирования с обратной связью обеспечивает поточное шифрование с обратной связью. Режим счетчика реализует поточное шифрование с использованием последовательности счетчика.

Пятый алгоритм предназначен для контроля целостности сообщений с помощью имитовставок. Шестая и седьмая группы обеспечивают совместное

шифрование и имитозащиту данных и ключей соответственно. Восьмая группа представляет алгоритм хеширования для преобразования данных произвольной длины в фиксированный размер.

Криптографическая стойкость алгоритма обеспечивается использованием 256-битного ключа, который исключает возможность полного перебора. Многократные итерации преобразований в 32 раунда создают выраженный лавинный эффект. Нелинейные преобразования через S-блоки значительно усложняют криптоанализ. Сложная ключевая схема генерации раундовых подключей обеспечивает дополнительную безопасность.

Преимущества стандарта СТБ 34.101.31-2011 включают соответствие современным требованиям криптографической защиты. Высокая эффективность реализации на программно-аппаратных платформах позволяет использовать его в различных системах. Стандарт обеспечивает как конфиденциальность, так и целостность данных. Совместимость с другими современными криптографическими стандартами позволяет интегрировать его в существующие системы защиты информации.

Стандарт предназначен для использования в системах защиты информации, требующих гарантированного уровня безопасности. Он соответствует национальным требованиям криптографической защиты данных и может применяться в государственных и коммерческих организациях. Использование данного стандарта обеспечивает надежную защиту информации от несанкционированного доступа и гарантирует сохранность конфиденциальных данных.

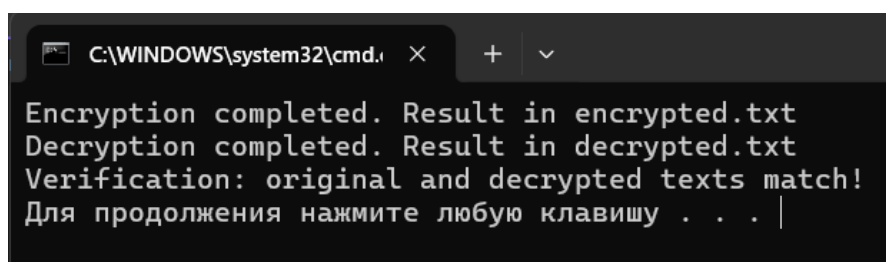
### 3 ХОД РАБОТЫ

В ходе выполнения лабораторной работы было разработано программное средство для криптографической защиты текстовых данных на основе алгоритма СТБ 34.101.31-2011. Реализация поддерживает режим простой замены (*ECB*) и обеспечивает как процесс шифрования, так и дешифрования информации.

Программный интерфейс реализован в консольном формате и автоматически выполняет полный цикл криптографических преобразований. Программа считывает исходные данные из текстового файла `input.txt`, выполняет их шифрование и сохраняет результат в файл `encrypted.txt`. Затем осуществляется дешифрование данных с сохранением результата в файл `decrypted.txt`.

Особенности реализации включают использование 256-битного ключа, генерируемого случайным образом для каждого сеанса работы. Программа обеспечивает выравнивание обрабатываемых данных до размера, кратного 16 байтам, что соответствует требованиям алгоритма. Для обработки текстовой информации используется расширенная кодировка ASCII, поддерживающая символы различных языков.

Наглядное представление работы программы и результаты выполнения операций демонстрируются на рисунке 3.1, где отображен процесс взаимодействия с программным средством. Также в процессе работы программы сравниваются полученный после расшифровки данных результат и исходный текст. Пользователь будет уведомлен в случае, если результаты совпадают или не совпадают.



```
C:\WINDOWS\system32\cmd.exe
Encryption completed. Result in encrypted.txt
Decryption completed. Result in decrypted.txt
Verification: original and decrypted texts match!
Для продолжения нажмите любую клавишу . . . |
```

Рисунок 3.1 – Консольный интерфейс программного средства

Исходный текст, подлежащий зашифровке, приведен на рисунке 3.2.

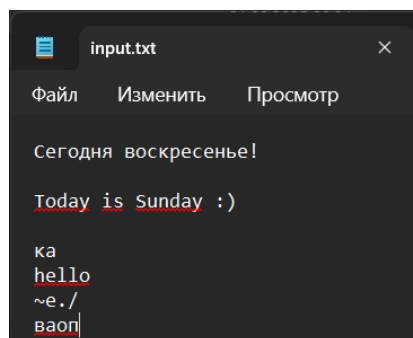


Рисунок 3.2 – Содержимое файла input.txt

Итоги криптографических преобразований приведены на рисунке 3.3.

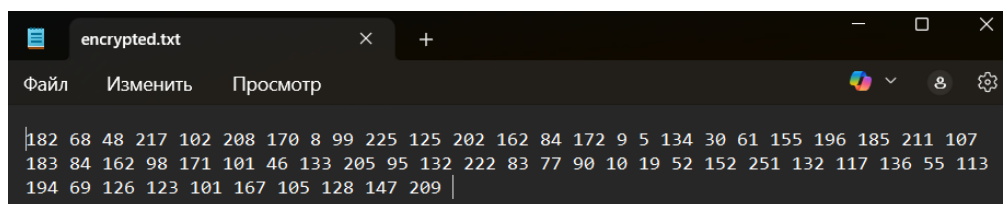


Рисунок 3.3 – Содержимое файла encrypted.txt

Расшифровка ранее зашифрованного текста приведена на рисунке 3.4.

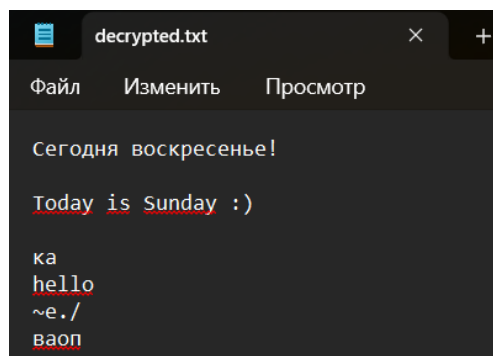


Рисунок 3.4 – Содержимое файла decrypted.txt

Верификация результатов подтверждает корректность работы алгоритма – исходный и дешифрованный тексты полностью совпадают, что свидетельствует о сохранении целостности данных при выполнении криптографических преобразований.

Разработанное программное средство успешно реализует алгоритм СТБ 34.101.31-2011 в режиме простой замены, демонстрируя практическую применимость для защиты конфиденциальной текстовой информации. Проведенные испытания подтверждают соответствие реализации требованиям стандарта и возможность ее использования для решения задач симметричного шифрования.

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы было успешно реализовано программное средство, реализующее алгоритм симметричного шифрования СТБ 34.101.31-2011. Разработанное решение демонстрирует работоспособность белорусского стандарта шифрования и его практическую применимость для защиты конфиденциальной информации. Реализация поддерживает основные операции: шифрование и дешифрование данных в режиме простой замены, обработку текстовых файлов, а также корректное выполнение выравнивания данных.

Проведенные испытания подтвердили корректность работы алгоритма – исходные текстовые данные полностью восстанавливаются после проведения полного цикла шифрования-дешифрования. Особое внимание было уделено обеспечению надежной работы с текстовой информацией на различных языках, включая обработку символов расширенной кодировки ASCII. Реализованная система автоматического выполнения операций обеспечивает удобство использования и может быть интегрирована в процессы обработки защищаемой информации.

Полученный опыт реализации криптографических алгоритмов представляет значительную ценность для понимания принципов современной защиты информации и может быть использован в дальнейшем для разработки более сложных систем информационной безопасности, соответствующих национальным стандартам Республики Беларусь.



# ПРИЛОЖЕНИЕ А

## (обязательное)

### Листинг программного кода

```
import random
import binascii
from typing import List, Union

class STB3410131:

    # H-table
    H_BOX = [
        0xB1, 0x94, 0xBA, 0xC8, 0x0A, 0x08, 0xF5, 0x3B, 0x36, 0x6D, 0x00,
0x8E, 0x58, 0x4A, 0x5D, 0xE4,
        0x85, 0x04, 0xFA, 0x9D, 0x1B, 0xB6, 0xC7, 0xAC, 0x25, 0x2E, 0x72,
0xC2, 0x02, 0xFD, 0xCE, 0x0D,
        0x5B, 0xE3, 0xD6, 0x12, 0x17, 0xB9, 0x61, 0x81, 0xFE, 0x67, 0x86,
0xAD, 0x71, 0x6B, 0x89, 0x0B,
        0x5C, 0xB0, 0xC0, 0xFF, 0x33, 0xC3, 0x56, 0xB8, 0x35, 0xC4, 0x05,
0xAE, 0xD8, 0xE0, 0x7F, 0x99,
        0xE1, 0x2B, 0xDC, 0x1A, 0xE2, 0x82, 0x57, 0xEC, 0x70, 0x3F, 0xCC,
0xF0, 0x95, 0xEE, 0x8D, 0xF1,
        0xC1, 0xAB, 0x76, 0x38, 0x9F, 0xE6, 0x78, 0xCA, 0xF7, 0xC6, 0xF8,
0x60, 0xD5, 0xBB, 0x9C, 0x4F,
        0xF3, 0x3C, 0x65, 0x7B, 0x63, 0x7C, 0x30, 0x6A, 0xDD, 0x4E, 0xA7,
0x79, 0x9E, 0xB2, 0x3D, 0x31,
        0x3E, 0x98, 0xB5, 0x6E, 0x27, 0xD3, 0xBC, 0xCF, 0x59, 0x1E, 0x18,
0x1F, 0x4C, 0x5A, 0xB7, 0x93,
        0xE9, 0xDE, 0xE7, 0x2C, 0x8F, 0x0C, 0x0F, 0xA6, 0x2D, 0xDB, 0x49,
0xF4, 0x6F, 0x73, 0x96, 0x47,
        0x06, 0x07, 0x53, 0x16, 0xED, 0x24, 0x7A, 0x37, 0x39, 0xCB, 0xA3,
0x83, 0x03, 0xA9, 0x8B, 0xF6,
        0x92, 0xBD, 0x9B, 0x1C, 0xE5, 0xD1, 0x41, 0x01, 0x54, 0x45, 0xFB,
0xC9, 0x5E, 0x4D, 0x0E, 0xF2,
        0x68, 0x20, 0x80, 0xAA, 0x22, 0x7D, 0x64, 0x2F, 0x26, 0x87, 0xF9,
0x34, 0x90, 0x40, 0x55, 0x11,
        0xBE, 0x32, 0x97, 0x13, 0x43, 0xFC, 0x9A, 0x48, 0xA0, 0x2A, 0x88,
0x5F, 0x19, 0x4B, 0x09, 0xA1,
        0x7E, 0xCD, 0xA4, 0xD0, 0x15, 0x44, 0xAF, 0x8C, 0xA5, 0x84, 0x50,
0xBF, 0x66, 0xD2, 0xE8, 0x8A,
        0xA2, 0xD7, 0x46, 0x52, 0x42, 0xA8, 0xDF, 0xB3, 0x69, 0x74, 0xC5,
0x51, 0xEB, 0x23, 0x29, 0x21,
        0xD4, 0xEF, 0xD9, 0xB4, 0x3A, 0x62, 0x28, 0x75, 0x91, 0x14, 0x10,
0xEA, 0x77, 0x6C, 0xDA, 0x1D
    ]

    BLOCK_SIZE = 16 # bytes
    WORD_SIZE = 4 # bytes
    ROUNDS = 8
    KEY_SIZE = 32 # bytes

    def __init__(self, key: Union[bytes, List[int]]):
        if len(key) != self.KEY_SIZE:
            raise ValueError(f"Key must be {self.KEY_SIZE} bytes long")

        # Convert key to words (32-bit numbers)
        key_words = [self.bytes_to_word(key[i:i+4]) for i in range(0,
len(key), 4)]
        self.round_keys = [key_words[i % 8] for i in range(56)]
```

```

def rotate_left(self, value, shift: int) -> int:
    bit_length = 32
    shift = shift % bit_length
    return ((value << shift) | (value >> (bit_length - shift))) &
0xFFFFFFFF

def word_to_bytes(self, word: int) -> List[int]:
    return [(word >> shift) & 0xFF for shift in [24, 16, 8, 0]]

def bytes_to_word(self, bytes_list: List[int]) -> int:
    return sum(byte << shift for byte, shift in zip(bytes_list, [24,
16, 8, 0]))

def reverse_word(self, word: int) -> int:
    bytes_list = self.word_to_bytes(word)
    bytes_list.reverse()
    return self.bytes_to_word(bytes_list)

def modular_subtract(self, x: int, y: int) -> int:
    return (x - y) & 0xFFFFFFFF

def modular_add(self, *values: int) -> int:
    result = 0
    for value in values:
        result = (result + self.reverse_word(value)) & 0xFFFFFFFF
    return self.reverse_word(result)

def h_box_substitution(self, byte: int) -> int:
    return self.H_BOX[byte]

def g_function(self, x: int, k: int) -> int:
    # Apply H-box to each byte
    substituted = self.bytes_to_word([self.h_box_substitution(byte)
for byte in self.word_to_bytes(x)])
    # Circular shift and return result
    return
self.reverse_word(self.rotate_left(self.reverse_word(substituted), k))

def encrypt_block(self, plaintext: List[int]) -> List[int]:
    if len(plaintext) != self.BLOCK_SIZE:
        raise ValueError(f"Block size must be {self.BLOCK_SIZE}
bytes")

    a, b, c, d = [self.bytes_to_word(plaintext[i:i+4]) for i in
range(0, self.BLOCK_SIZE, 4)]

    # 8 rounds
    for round_num in range(self.ROUNDS):
        b ^= self.g_function(self.modular_add(a,
self.round_keys[7*round_num + 0]), 5)
        c ^= self.g_function(self.modular_add(d,
self.round_keys[7*round_num + 1]), 21)

        a =
self.reverse_word(self.modular_subtract(self.reverse_word(a),

self.reverse_word(self.g_function(self.modular_add(b,
self.round_keys[7*round_num + 2]), 13))))

        e = self.g_function(self.modular_add(b, c,
self.round_keys[7*round_num + 3]), 21) ^ self.reverse_word(round_num + 1)

```

```

        b = self.modular_add(b, e)
        c = self.reverse_word(self.modular_subtract(self.reverse_word(c),
self.reverse_word(e)))
        d = self.modular_add(d, self.g_function(self.modular_add(c,
self.round_keys[7*round_num + 4]), 13))

        b ^= self.g_function(self.modular_add(a,
self.round_keys[7*round_num + 5]), 21)
        c ^= self.g_function(self.modular_add(d,
self.round_keys[7*round_num + 6]), 5)

        # Word permutations
        a, b = b, a
        c, d = d, c
        b, c = c, b

    # Result
    result = []
    for word in [b, d, a, c]:
        result.extend(self.word_to_bytes(word))

    return result

def decrypt_block(self, ciphertext: List[int]) -> List[int]:
    if len(ciphertext) != self.BLOCK_SIZE:
        raise ValueError(f"Block size must be {self.BLOCK_SIZE}
bytes")

    a, b, c, d = [self.bytes_to_word(ciphertext[i:i+4]) for i in
range(0, self.BLOCK_SIZE, 4)]

    # 8 rounds
    for round_num in reversed(range(self.ROUNDS)):
        b ^= self.g_function(self.modular_add(a,
self.round_keys[7*round_num + 6]), 5)
        c ^= self.g_function(self.modular_add(d,
self.round_keys[7*round_num + 5]), 21)

        a = self.reverse_word(self.modular_subtract(self.reverse_word(a),
self.reverse_word(self.g_function(self.modular_add(b,
self.round_keys[7*round_num + 4]), 13))))

        e = self.g_function(self.modular_add(b, c,
self.round_keys[7*round_num + 3]), 21) ^ self.reverse_word(round_num + 1)

        b = self.modular_add(b, e)
        c = self.reverse_word(self.modular_subtract(self.reverse_word(c),
self.reverse_word(e)))
        d = self.modular_add(d, self.g_function(self.modular_add(c,
self.round_keys[7*round_num + 2]), 13))

        b ^= self.g_function(self.modular_add(a,
self.round_keys[7*round_num + 1]), 21)
        c ^= self.g_function(self.modular_add(d,
self.round_keys[7*round_num + 0]), 5)

    # Reverse word permutations

```

```

        a, b = b, a
        c, d = d, c
        a, d = d, a

    # Result
    result = []
    for word in [c, a, d, b]:
        result.extend(self.word_to_bytes(word))

    return result

def char_to_extended_ascii(ch):
    code = ord(ch)
    return code if code < 140 else code - 900

def extended_ascii_to_char(code: int) -> str:
    return chr(code) if code < 140 else chr(code + 900)

def generate_key() -> bytes:
    return bytes(random.randint(0, 255) for _ in range(32))

def main():
    # Read input file
    try:
        with open("input.txt", "r", encoding='utf-8') as file:
            text = file.read()
    except FileNotFoundError:
        print("Error: input.txt file not found")
        return
    except UnicodeDecodeError:
        print("Error: cannot read file in UTF-8 encoding")
        return

    original_text = text

    count = 0
    while len(text) % 16:
        text += '0'
        count += 1
    encrypted_result = []
    decrypted_result = []

    for i in range(len(text) // 16):
        arr_text = [char_to_extended_ascii(item) for item in text[16 * i:
16 * (i + 1)]]

        random_bytes = bytes([random.randint(0, 255) for _ in range(32)])
        hex_str = random_bytes.hex()
        key = list(binascii.unhexlify(hex_str))
        my_stb = STB3410131(key)
        encrypted = my_stb.encrypt_block(arr_text)
        encrypted_result.extend(encrypted)
        decrypted_result.extend(my_stb.decrypt_block(encrypted))

    encrypted_result = encrypted_result[:len(encrypted_result) - count]
    decrypted_result = decrypted_result[:len(decrypted_result) - count]
    with open("encrypted.txt", "w", encoding='utf-8') as f:
        for item in encrypted_result:

```

```

        f.write(str(item) + ' ')
    print("Encryption completed. Result in encrypted.txt")

    decrypted_text = ""
    with open("decrypted.txt", "w", encoding='utf-8') as f:
        for item in decrypted_result:
            char = extended_ascii_to_char(item)
            decrypted_text += char
            f.write(char)

    print("Decryption completed. Result in decrypted.txt")

    if original_text == decrypted_text:
        print("Verification: original and decrypted texts match!")
    else:
        print("Warning: original and decrypted texts do not match!")

if __name__ == '__main__':
    main()

```

# ПРИЛОЖЕНИЕ Б

## (обязательное)

### Блок-схема алгоритма, реализующего программное средство

